Massachusetts Institute of Technology
Artificial Intelligence Laboratory

AI Memo 384 December 1976 Logo Memo 31

AI Based Personal Learning Environments: Directions for Long Term Research

Ira P. Goldstein and Mark L. Miller

The application of artificial intelligence (AI) techniques to the design of personal learning environments is an enterprise of both theoretical and practical interest. In the short term, the process of developing and testing intelligent tutoring programs serves as a new experimental vehicle for exploring alternative cognitive and pedagogical theories. In the long term, such programs should supplement the educational supervision and guidance provided by human teachers. This paper illustrates our long term perspective by a scenario with a hypothetical tutoring system for elementary graphics programming.

This report is a revised version of the first half of AI Working Paper 122 (Logo Working Paper 50). It describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. This research was supported in part by the National Science Foundation under grant C40708X, and in part by the Division for Study and Research in Education, Massachusetts Institute of Technology.

Table of Contents

Acknowledgements

Thanks are due to H. Abelson and H. Peelle for carefully criticizing an earlier version of this paper.

1. Introduction

> ...If [the science of education] intends to limit itself, in conformity with the positivist conception of science, to a simple investigation into facts and laws, without claiming to explain what it states, then naturally there is no need whatever for a connection with psychology.... But if experimental pedagogy wishes to understand what it is doing and to complete its observations with causal interpretations or "explanations", it is obvious that it will have to employ a precise psychology...
>
> [Piaget 1972]

The design of AI based tutoring programs represents a new paradigm for educational research. This paradigm involves a unique marriage of methods and goals from artificial intelligence, psychology, traditional pedagogy, and computer science.

1. The *AI goal* is to explore powerful computational (AI) theories of learning and problem solving. The method is to design, implement and test programs that embody these theories.

2. The *psychological goal* is to account for the knowledge states and learning strategies of individual students. The method is to analyze protocols of students interacting with precisely controlled tutoring environments.

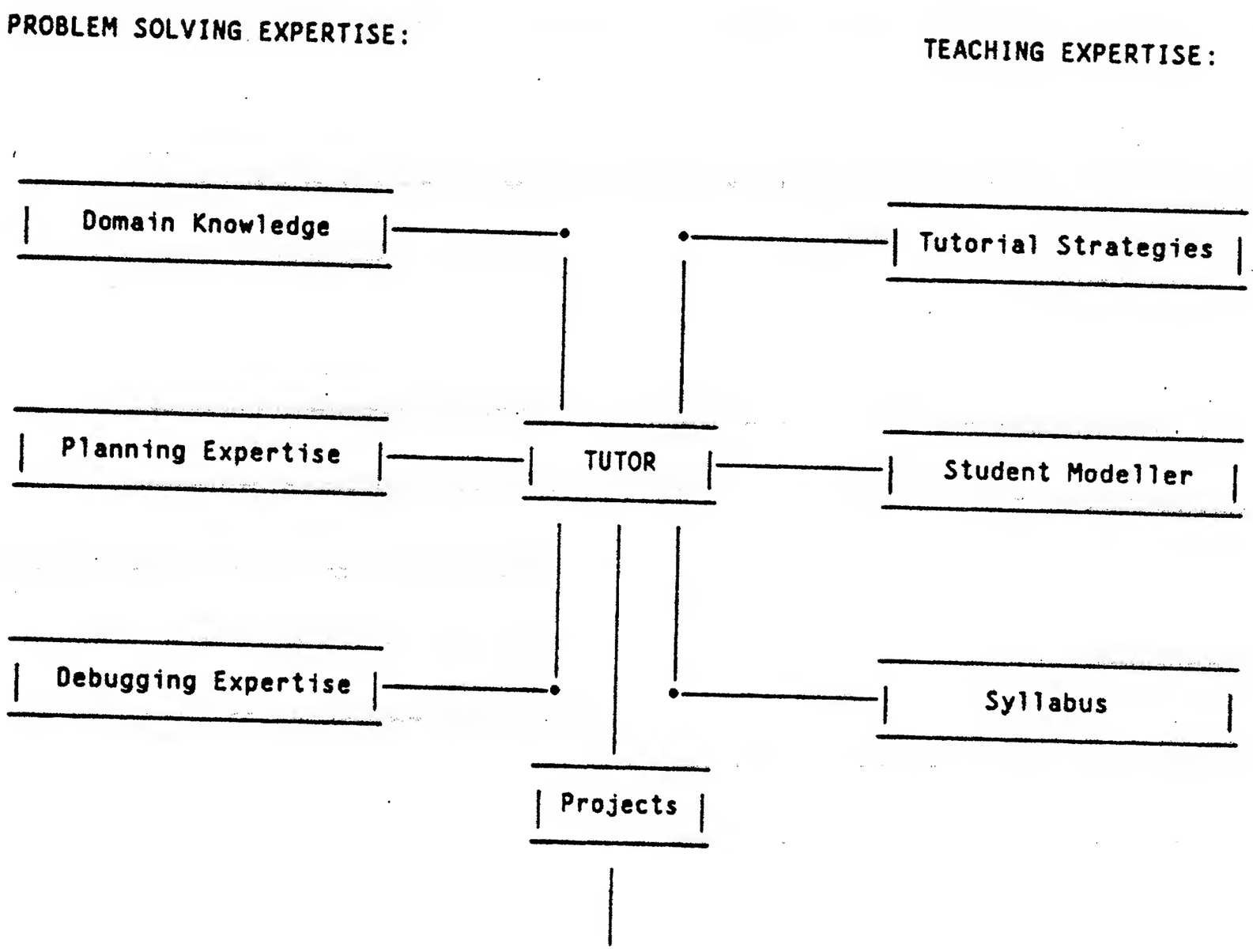
3. The *pedagogical goal* is to experiment rigorously with different tutorial strategies. The method is to systematically vary the pedagogical strategies employed by the tutor.

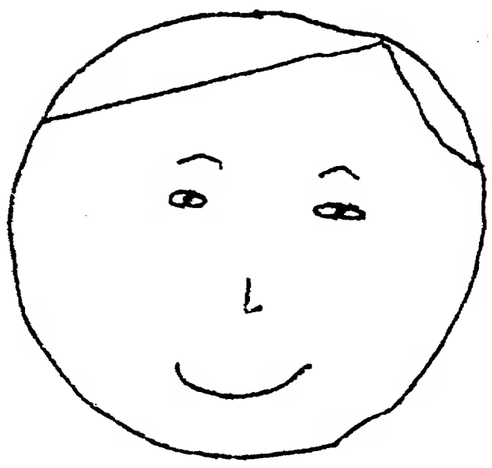
4. The *computer science goal* is to articulate the design principles for programs of this nature and level of complexity. The method is to construct and debug such systems.

This paper introduces the direction of our research by means of a scenario involving Sherlock, an *imaginary* computer tutor.[1] {Figure 1} provides a tentative block diagram for Sherlock. Understanding the theory and technology that will make such computer tutors possible is our ultimate goal; developing the individual components constitutes our current research focus. Since

An Intelligent Tutoring System

(Idealized Block Diagram)





{Figure 1}

each component is a significant undertaking in its own right, capabilities at the level this scenario proposes are not imminent. But if the highly restrictive teaching programs provided by traditional computer aided instruction (CAI) are to be superseded, then the design of learning environments which incorporate computational models of both the student and the teacher is essential.

This paper is not a technical exposition: its sole purpose is to convey our sense of the direction in which CAI *ought* to be headed. It does not address the objection that tutors such as Sherlock may be too far beyond the state of the art to be a profitable research focus. Evidence that research into the design of AI based learning environments provides a powerful medium for exploring computational theories of cognition and pedagogy is described elsewhere. (See, for example, [Brown et al. 1974], [Collins et al. 1975], [Burton & Brown 1976], [Goldstein & Miller 1976b], [Miller & Goldstein 1976a,b,c,d].)

2. Scenario With a Hypothetical Logo Tutor

> ...leave the students as much freedom and initiative as possible under existing teaching conditions. Pressed for time, the mathematics teacher is often tempted to sin against ... the *principle of active learning*. He may hurry to the solution of a problem without leaving enough time for the students to put the problem to themselves in earnest. He may name a concept or formulate a rule too soon, without sufficient preparation by appropriate material, before the students can feel the need for such a concept or rule. He may commit the celebrated mistake of *deus ex machina*: he may introduce some device (for instance, a tricky auxiliary line in a geometric proof) which leads to the result all right, but the students cannot see for their life how it was humanly possible to discover such a trick which appeared right out of the blue.
>
> [Polya 1965]

This section illustrates, in the form of a hypothetical scenario, the nature of Sherlock, an intelligent tutor which we are designing and plan to implement. The tutor would operate in the domain of Logo turtle geometry. The *turtle* is a *graphics cursor* capable of drawing on a computer display. The turtle has a *state*, which consists of its location, its heading, and whether its pen is up or down. Primitives in the Logo programming language (FORWARD, RIGHT, PENUP, PENDOWN) alter these attributes of the state vector independently. The task of the student is to define procedures that draw his or her intended pictures.

We have chosen this domain for our initial example for the same reason that it is used by the Logo Project: it is an excellent environment for revealing problem solving methods [Papert 1971a,b; 1973]. Both the general medium of programming and the particularly concrete nature of graphics tasks encourage this. Furthermore, there has been a great deal of work with students in this domain. Many case studies of individual students are available (e.g., G. Goldstein [1972], Okumura [1972], Solomon [1976b]), and several styles of teaching have been investigated (e.g., Austin [1976], Hildreth [1975], Solomon [1976a]). However, though introductory Logo programming will be our initial domain for designing a tutor, we believe that many attributes of the Sherlock tutor, including its general problem solving knowledge, the methods governing the construction of a student model, and the various pedagogical techniques directing the tutorial process, will be applicable to the design of tutors for other areas of study.

The tutor's practical goal is to guide the student through introductory Logo programming projects. Conceptually, its objectives are to advise the student concerning the use of various problem solving techniques. These include:

1. alternative planning methods such as sequential planning, in which a problem is divided into separate pieces which are solved independently; and recursive planning, in which a problem is reduced to a simpler version of itself until a directly solvable case is reached;

2. debugging knowledge for diagnosing the underlying cause of a bug, understanding the typical failings to which various plan types are susceptible, and handling complex situations involving multiple bugs;

3. anticipatory strategies for ensuring program extensibility and ease of debugging, such as adequate documentation and modular design;

4. the importance of carefully describing the problem, the solution plan, the finished program and the relations between them.

The scenario raises many complex questions of tutorial strategy, such as how frequently to interrupt the student with advice, and under what circumstances to allow the student to proceed even though a blind alley is being pursued. Furthermore, the scenario presumes powerful facilities which are peripheral to this essay: for example, a sketch recognizer for tablet input, and a sophisticated natural language front end. The reader is asked to postpone concern with these important problems. Since it would be impossible to address so many issues in a single paper, we concentrate, instead, on a single aspect: the fashion in which an understanding of *planning and debugging* guides the tutor's interactions with the student.

2.1. Defining

Hello.

Welcome to Sherlock, the tutorial Logo monitor! What is your name?

Debbie Smith.

Hello, Debbie. Good to see you again!

Here it has been assumed that the student has used the learning environment previously. Consequently, Sherlock is able to read in files containing the programs which she has written, the history of previous sessions, and a detailed model of her current knowledge and skills.

I have been instructed to provide maximum tutoring.[2] I will make suggestions whenever I think that they might be helpful. Feel free to ask questions at any time. Also, if you tell me a little about what you are trying to do, I can be more helpful.

A practical system would require flexible natural language input which allowed the student to ask questions at any time. A simple parser with a small grammar, however, might be adequate for a workable prototype.

I am going to draw a napoleon.

I don't think I have ever seen a napoleon. Perhaps you could draw one on the tablet, just to give me the idea.

Ok.

Ideally, Sherlock would have a module for analyzing simple sketches drawn on a tablet. This would obviate the need for the student to describe the intended picture in a formal model language. Instead, the system would have a large database of knowledge about the pictures commonly encountered in beginning projects represented as predicate models. It would be the tutor's responsibility to match a model to the sketch.[3] See {Figure 2} for an example of such a picture model for a stick figure and {Figure 3} for the student's tablet input.

Oh, I see, napoleon is a stick man, only with a funny head!

The sketch understander would identify this particular sketch as an instance of the STICKMAN model, except that the HEAD is unusual.

That's his hat!

Oh! Ok.

Predicate Model* For A Stickman

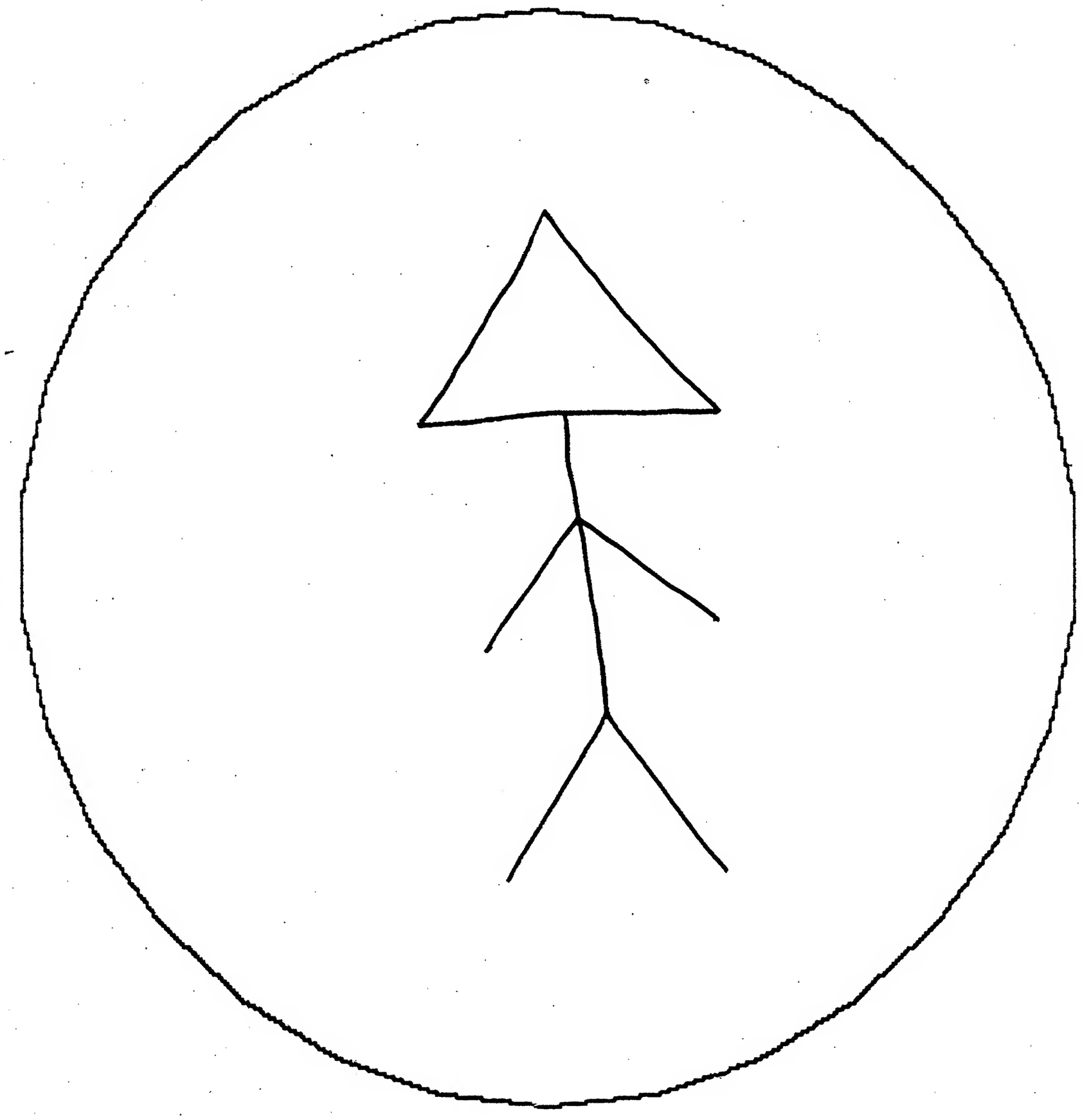
```
MODEL MAN
M1 PARTS HEAD BODY ARMS LEGS
M2 EQUITRI HEAD
M3 LINE BODY
M4 V ARMS, V LEGS
M5 CONNECTED HEAD BODY, CONNECTED BODY ARMS, CONNECTED BODY LEGS
M6 BELOW LEGS ARMS, BELOW ARMS HEAD
END


MODEL V
M1 PARTS L1 L2
M2 LINE L1, LINE L2
M3 CONNECTED L1 L2 (VIA ENDPOINTS)
END


MODEL EQUITRI
M1 PARTS (SIDE 3) (ROTATION 3)
M2 FOR-EACH SIDE (= (LENGTH SIDE) 100)
M3 FOR-EACH ROTATION (= (DEGREES ROTATION) 120)
M4 RING CONNECTED SIDE
END
```

* (The predicate model language is an external representation which would be translated to an internal format very similar to first order predicate calculus.)

(Figure 2)

Intended Napoleon

(Hand Drawn on the Tablet)



{Figure 3}

Sherlock would modify its representation for the NAPOLEON version of STICKMAN to reflect this additional information.

Sherlock would now be ready to assist Debbie in defining a program to satisfy the NAPOLEON predicate model. We envision Sherlock as being capable of providing more than one overall style of assistance. One mode would be primarily backward-looking, allowing the student to proceed without a great deal of help, but occasionally interrupting to ask the student for explanation or clarification of the reasoning involved in previous events. The requested explanations serve two functions: they cause the student to reflect on and become articulate about the problem solving process, and they provide guidance to internal modules of Sherlock (such as the student modeller) when the purpose of a type-in is ambiguous or obscure.

Initially in its backward-looking mode, Sherlock remains silent, as Debbie types:

```
right 90
forward 50
left 120
forward 100
left 120
forward 100
left 120
forward 500
```

{Figure 4}

Oh, no! Erase that last forward.

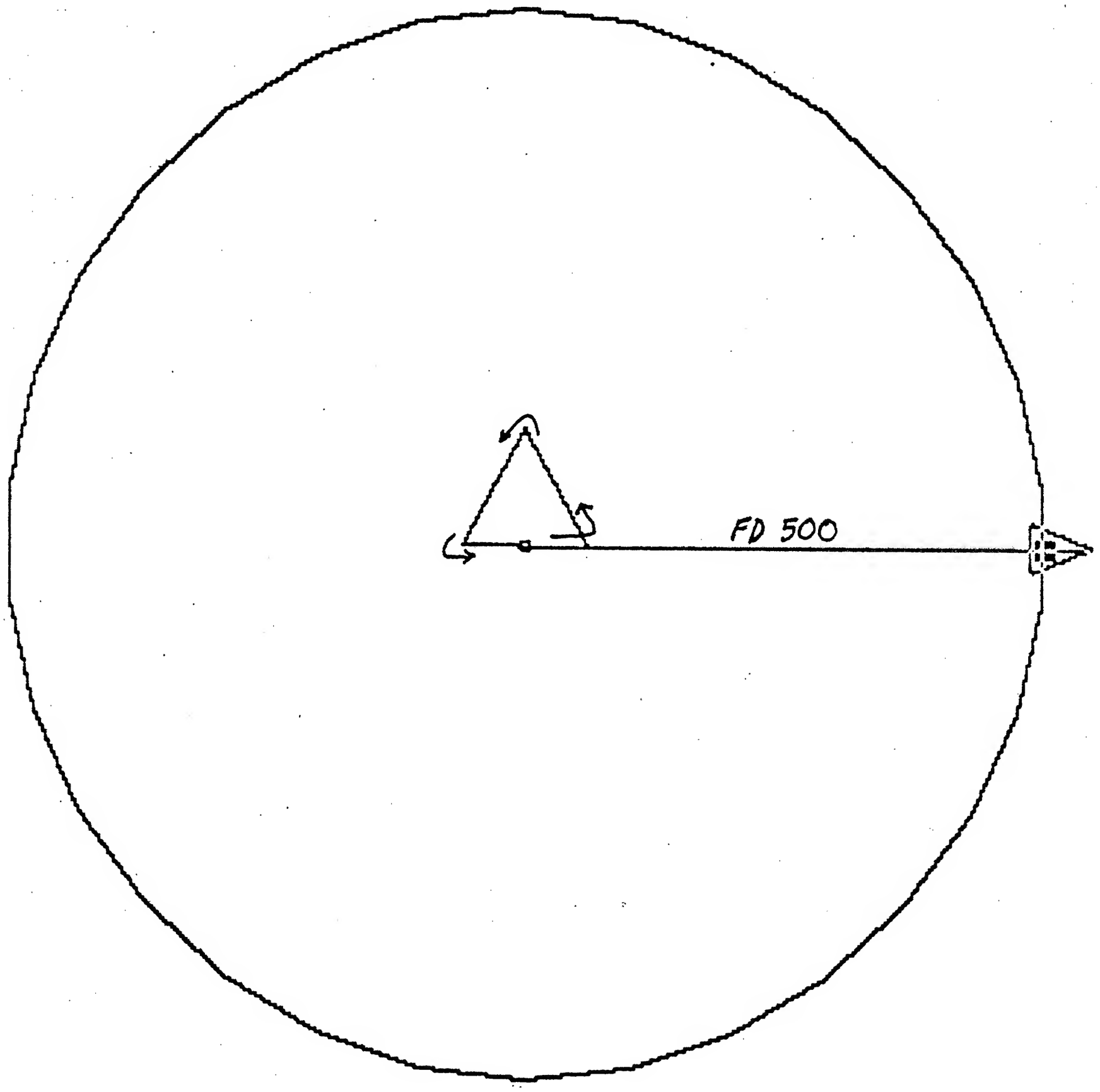
Ok. "Forward 500" has been erased. Do you wish to define the other lines as a procedure?

Ordinarily, in this backward-looking style, the tutor would intervene only upon request or when an error has occurred. It is probably at such points that the student will be most amenable to suggestions. In this case, the tutor's intervention immediately follows the occurrence of a typing bug. The tutor's subsequent question serves to remind the student of the importance of using subprocedures.

A more forward-looking style of tutoring should also be possible, in which the tutor actively prompts the student. Such a mode takes more of the initiative away from the student, however, and should be used with discretion. Here, the tutor could have decided to intervene

Debbie's Initial Type-ins

(oh, no! erase that last fd!)



{Figure 4}

sooner, provided that there was sufficient cause. On the basis of running its simulation model of Debbie, the tutor might have expected that she would *not* use subprocedures, and hence would encounter too many bugs attempting the napoleon problem on her own. Even so, with its innate bias toward active learning, Sherlock would hesitate to interfere. However, for illustrative purposes, assume that in this case the Sherlock tutor did enter its forward-looking mode. Hence, immediately following the sketch recognition sequence, but before Debbie began to type in code, the tutor might have asked:

Do you have a plan?

Well, sort of. I thought I would start with his hat and work downward.

This response suggests that Debbie has mastered a particular kind of "sequential" plan in which the main steps are achieved in an order which simplifies interfacing. The ordering strategy is based on observing the domain-dependent fact that BELOW is a transitive predicate. These inferences regarding Debbie's planning knowledge would be recorded in the student model. The new information would cause the tutor to shift its emphasis to a different topic: in this case, the modular design of the individual main steps. Once again, for expository purposes, the tutor is shown taking a too-strong initiative.

Are you going to use any subprocedures?

No.

Hmmm. I notice that you already have a program to draw a triangle. Perhaps you could use that?

The tutor would try to build upon previously solved problems, generalizing and extending them. One of the ways in which this "evolutionary" planning strategy can be manifested is by using existing programs as subprocedures in the new project.

In order to highlight a crucial pedagogical issue, in this instance we have portrayed Sherlock providing questionable advice: TRIANGLE contains a bug in that it does not halt. The critical issue is whether this sort of tactic is sound, in that it increases the frequency with which the student encounters *pedagogically valuable bugs*, i.e. difficulties whose correction provides valuable learning experience.[4]

Ok. That can be his hat. Here is the definition.

```
to napoleon
triangle
back 50
left 120
forward 100
back 100
left 30
forward 100
back 100
right 30
forward 100
right 30
forward 100
back 100
left 60
forward 100
end
```

Debbie has now defined her program. Her next action is to run the program.

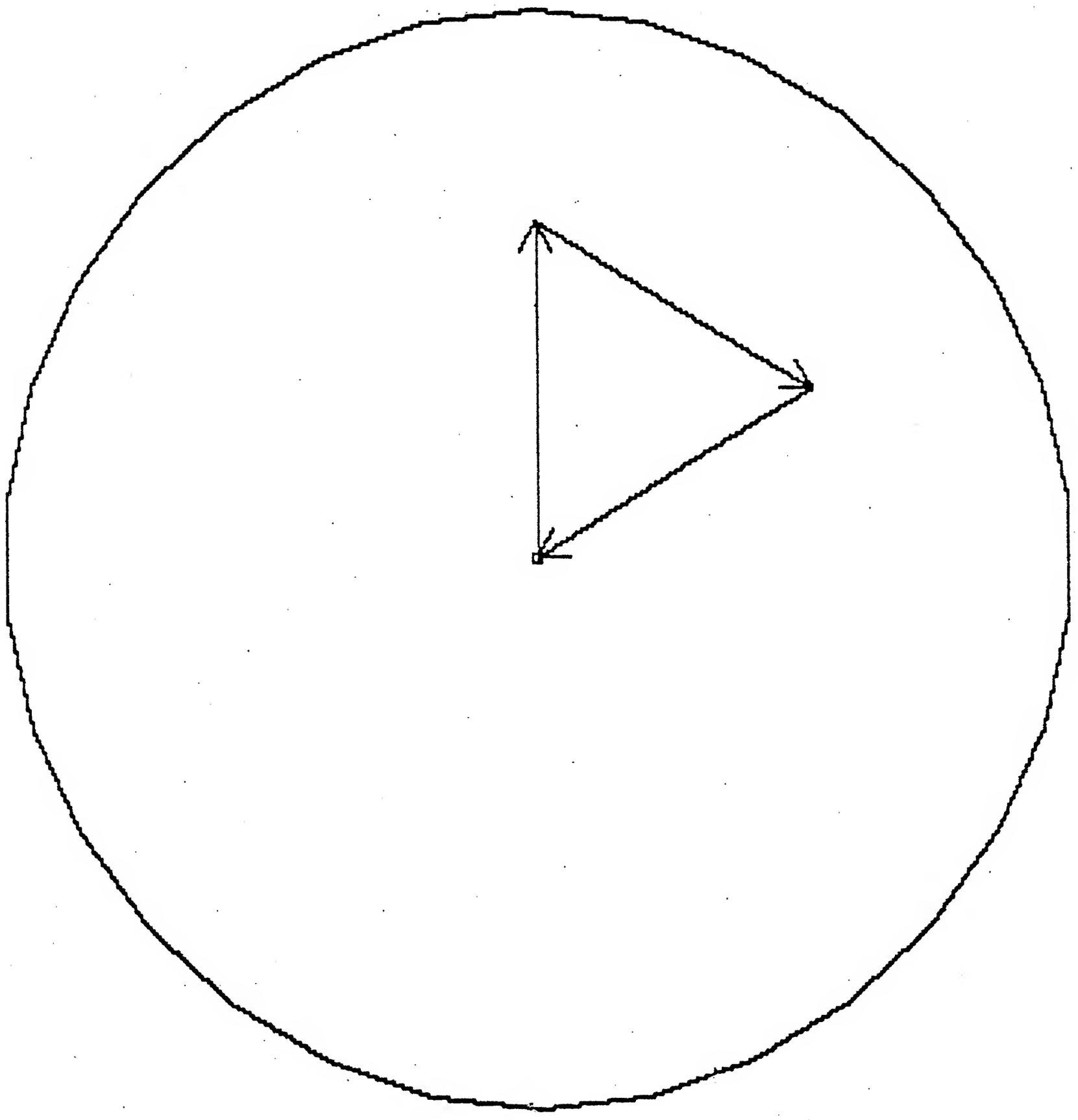
napoleon {Figure 5}

Here, the tutor would call upon an expert program understanding module called Mycroft [Goldstein 1974] to annotate the performance of Debbie's program, analyze the unspoken purposes of the individual lines of code and diagnose bugs.[5] Mycroft would notice several bugs in the resulting picture, with respect to the napoleon model.

Note that Debbie's code employs few subprocedures, and with its bugs is not easy to understand. This might be an appropriate point for explanatory requests by the tutor. Debbie could be asked, for example, to explain the purposes of certain lines of her code, or to indicate which of several plausible interpretations she intended. The questions would serve as *confirmation* of the Mycroft analysis. Hence the questions would be at an appropriately high level -- not tedious -- and might unobtrusively help Debbie to pinpoint some of her bugs.

Napoleon, Version 1

(Triangle Forever)



{Figure 5}

2.2. Debugging

Mycroft's analysis of the bugs in Debbie's program would indicate to Sherlock which debugging strategies might be appropriate. Usually, the tutor would not immediately criticize the student's program. However, the expectations established by Mycroft's analysis would serve to guide its response to possible student questions. Suppose that Debbie partly diagnoses the cause of one of the bugs on her own.

It has a bug! It's just drawing the same old triangle, over and over. How come?

Well, let's see. Would you like to look at your TRIANGLE code?

The tutor is shown here guiding the student's search for the underlying cause of the bug with an extremely leading question. (It suggests which subprocedure has the bug.) This would be reasonable in a situation in which the tutor does not believe that Debbie could localize the bug without help. But suppose Sherlock believed that Debbie *could* localize the bug herself. An alternative scenario might continue as follows:

Well, let's see. How could we find such a bug?

I give up.

Well, what is the first thing your procedure does?

I don't know.

What is the first subprocedure called by napoleon?

Returning to the main dialogue, one serious problem with NAPOLEON is that TRIANGLE is a non-halting subprocedure. Even though the tutor has supplied a strong hint, it has not guaranteed that TRIANGLE is the faulty routine, nor has it explained *why* the code fails to halt. It *has* attempted to encourage the student to employ the debugging technique of checking the suspicious code independently of the remainder of the program.

Debugging a suspicious code segment "in private" is an important debugging strategy,

which seeks to simplify the analysis by excerpting the questionable code and examining it (and perhaps executing it) in isolation. Although often helpful, this debugging technique can be insufficient if the bug is due to some unexpected interaction between the subprogram and the global run-time environment. Nevertheless, it is an effective initial step toward isolating the problem. Suppose Debbie accepts the suggestion.

Yes. Print out triangle.

```
TO TRIANGLE
FORWARD 100
RIGHT 120
TRIANGLE
END
```

Do you see the problem?

Yes. Isn't there some way I can make it stop?

Understanding this question (even after it had been parsed into a formal query language) would require considerable guidance from expectations about the application domain (i.e., turtle geometry), programming languages in general, and prior knowledge of the student. Here, the tutor encourages applying another evolutionary strategy.

Yes. Do you remember how we did it for SQUARE? Perhaps you could use that as a pattern to work from.

Oh, right, ok. Redefine triangle to be:

```
to triangle
repeat 3 triangleside
end
```

```
to triangleside
forward 100
right 120
end
```

Now run napoleon again.

Here it is. {Figure 6}

Debbie's ability to make the analogy, and incorporate the appropriate modifications of the SQUARE pattern, is evidence that she understands the iterative programming construct. This should be noticed by the student modelling component of Sherlock.

2.3. Planning

The tutor might continue to guide Debbie through debugging her NAPOLEON procedure as originally designed. But the purpose of Logo is not merely to achieve working programs. Debbie's procedure was developed using a "sequential" plan. But it used "in-line code" to implement main steps which -- according to well-established precepts of programming style -- should have been implemented as calls to separate subprocedures. So in this case the tutor is depicted recommending that Debbie *revise* her original plan, to encourage modularity and facilitate debugging.

It's still not right.

I know. It might be easier to debug if you used subprocedures more. What about having a separate procedure for each part: the hat, the neck, the arms, the body, and the legs?

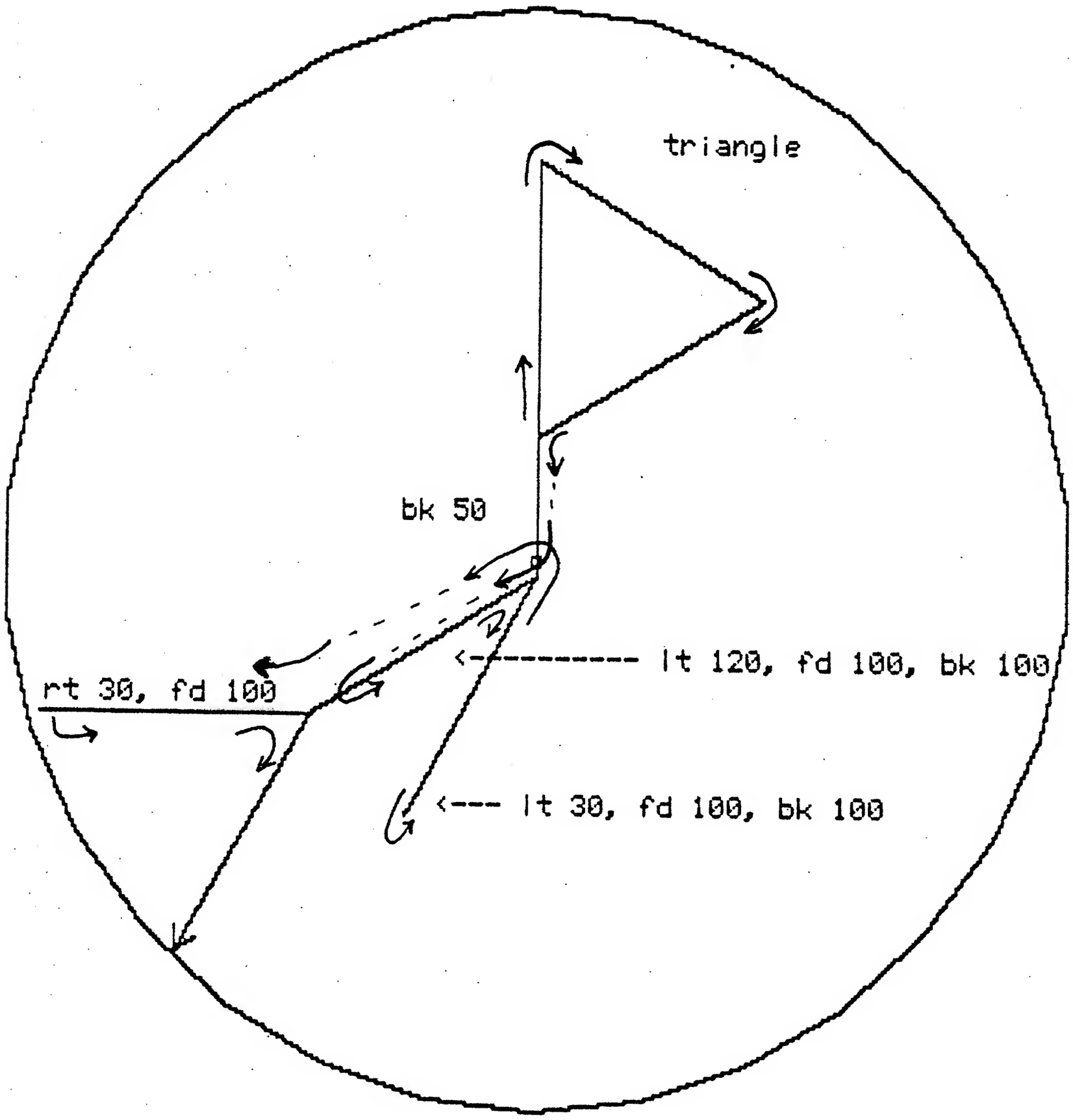
I guess it wouldn't hurt.

I think it would help -- then you could debug each one separately. When they were all working, you could put them all together, and there would probably be fewer problems.

Ok. Here is the new napoleon program:

Napoleon, Version 2

(Without Subprocedurization)



{Figure 6}

```
to napoleon
triangle                    !his hat!
neck
arms
body
legs
end
```

Considerable progress! Still, it would be more modular if the hat, accomplished by the call to TRIANGLE, were implemented as a separate procedure -- called (say) "HAT." This is because the name "HAT" describes *what* the procedure is to accomplish; whereas the name "TRIANGLE" describes *how* the hat is to be accomplished, information that the calling procedure (i.e., NAPOLEON) need not know. The tutor would normally not intervene here, though, since the student has probably been interrupted too frequently already. For illustrative purposes, the tutor is shown as overly enthusiastic.

That seems a lot better. But perhaps you should make the hat a separate procedure from TRIANGLE. That way, if you want to change it to a different kind of hat, you won't mess up your triangle.

No, I don't think I'll want other kinds of hats. And if I did, I could always change it later.

Very well.

Even in "maximal intervention mode," the tutor should eventually relent! When the student persists, it is wise to let the difficulty run its course. Further insistence by the tutor on its notion of what is proper (in this case, modularity) will seem arbitrary. If the advice is sound, the probability is that subsequent events will reinforce the tutor's suggestions.

Shall we continue working on the other subprocedures?

yes:

```
to neck
back 50
end
```

```
to arms
left 120
forward 100
back 100
right 30
forward 100
back 100
left 60
end

to body
forward 100
end

to legs
left 120
forward 100
back 100
right 30
forward 100
back 100
left 60
end
```

In the course of analyzing these programs, the tutor (via Mycroft) should notice that ARMS and LEGS are identical. This suggests that the student has missed a significant generalization. Since it has been assumed that Sherlock is currently emphasizing modularity, the matter would brought to the student's attention immediately.

Do you see anything interesting about the procedures for the arms and the legs?

Now that you mention it, as a matter of fact they are the same.

Perhaps you could improve your code by having a subprocedure which they both use.

Ok. Call it P1. Have both arms and legs just be calls to P1.

Fine. Perhaps you can think of a better name for "P1"? Otherwise, you might forget what it does. Does the shape it draws suggest any names to you?

It looks like an upside down "V". Maybe I'll call it "VEE". Now run napoleon.

Probably the tutor, as just portrayed, has gone too far. The story might have continued differently:

Why don't you write the program yourself?[6]

. . .

However, continuing as before, with a very cooperative student: NAPOLEON now draws {Figure 7}.

The hat's in the wrong place. Edit triangle.

This bug provides a more convincing demonstration of the previous point regarding modular design than the tutor's admonitions ever could.

TRIANGLE is also called by TREE, which you were working on before. If you change TRIANGLE, it could affect TREE. Are you sure you want to do that?

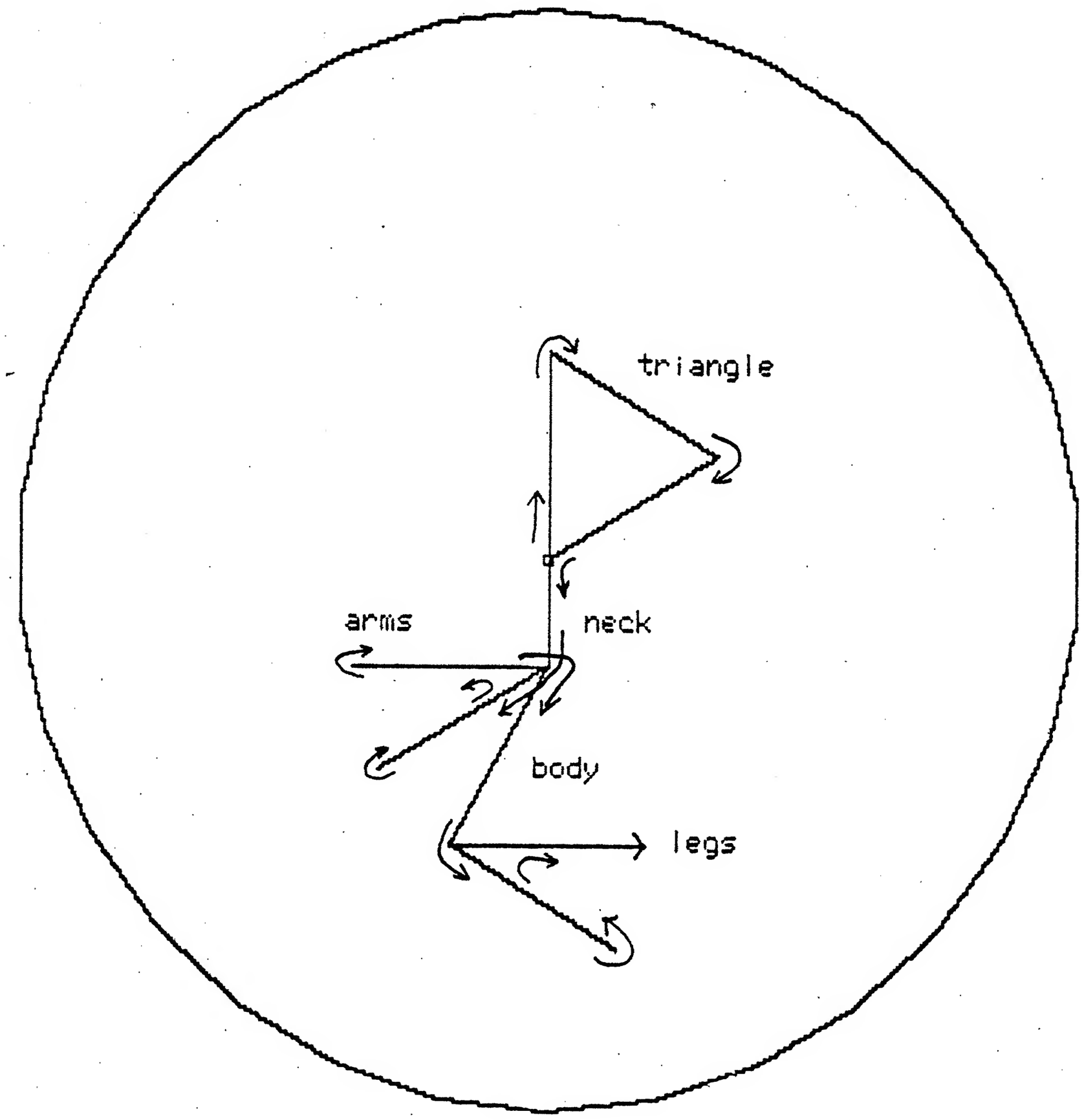
Note that TREE was also affected by the previous edit to TRIANGLE, which converted it from a non-halting recursive procedure to an iterative procedure with an implicit stop rule. However, this change was entirely beneficial, fixing a known problem in a partially finished FOREST {Figure 8}. Perhaps the tutor should be considered remiss, in not having pointed out this beneficial side-effect of using subprocedures immediately. We supposed that Sherlock chose to defer mentioning it, to avoid disrupting Debbie's train of thought. The matter could still be brought up later, during a recapitulation of the session.

All right. Change napoleon to call hat. Make hat call triangle for now.

(NAPOLEON edited. HAT defined.)
-- A wise decision!

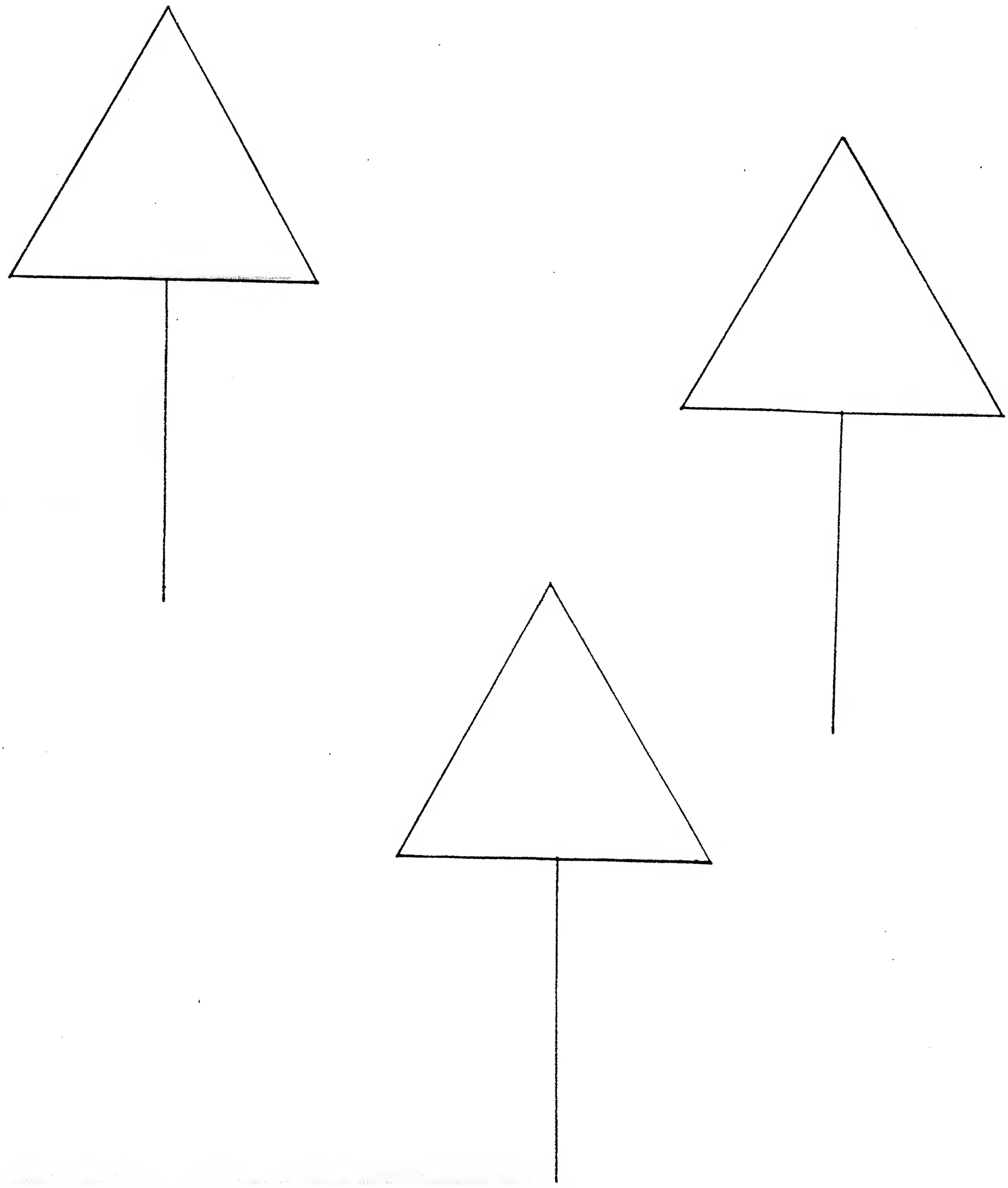
Napoleon, Version 3

(With Subprocedurization)



{Figure 7}

The Intended Forest



{Figure 8}

Debbie has probably learned the lesson. But she is about to make another very common mistake: approaching a debugging situation without a coherent strategy. Unless she demonstrates an understanding of the underlying causes of each error, her proposed edits are apt to be haphazard or even counterproductive.

Edit hat. Add "right 90" at the beginning.

Do you understand what is wrong with it?

Not exactly. It's just wrong.

Things could have been different if tutoring were completely switched off. For example:

HAT edited.

Run Napoleon. ...
Its worse than ever! I'll never get it. I think I'll quit for today.

Continuing where we left off:

Does it draw a triangle, like it is supposed to?

Yes.

Well then, the problem is probably in the setup or cleanup for the hat. Is the bug that it is starting in the wrong place, or stopping in the wrong place?

It finishes up in the wrong place. Besides, it's tilted.

Then perhaps what you need is a setup step before calling it, or a cleanup step after calling it, rather than any changes to the hat itself?

Currently, the hat is a "state transparent" subprocedure: it always ends in exactly the same state in which it begins. (This is analogous to "interrupts" in operating systems.) This program organization is often preferable to one in which the ending state of each routine is hand-

tailored to correspond to the desired starting state of its successor. In the latter case, rearrangement of the order of accomplishing parts leads to a necessity to modify main steps, even though they have previously been shown to accomplish their specified goals. But when state transparent main steps are interspersed with interface steps in the calling procedure, such changes are more easily made. This is precisely the sort of knowledge which the tutor is expected to convey.

> **Ok. I'll put in a cleanup step after the call to hat. Let's see. After hat, the turtle is facing north. Also, it's at a vertex of the triangle, but it should be in the middle of the bottom. So these steps are the cleanup for after the call to hat:**
>
> ```
> right 90
> forward 50
> right 90
> ```
>
> **Now its facing down, ready to draw the neck.**

An important component which would be constantly operating behind the scenes is the "purpose oriented editor" (cf. [Miller & Goldstein 1976c]). Such an editing facility would accept requests, such as those shown here, in which the location of the edit is specified in terms of its meaning: for example, as a "cleanup step" relative to some "main step" and not in terms of (the usual) line numbers. Such a very high level editor avoids the necessity of communicating with the computer in an excessively syntactic fashion. Learning editing should not interfere with the primary task of learning to solve problems effectively.

> ***Ok. Do you realize that NECK is defined as:***
>
> ```
> TO NECK
> BACK 100
> END
> ```

The purpose oriented editor would routinely check for undesirable side effects of edits. These usually result from forgotten assumptions about the specifications of interacting subprograms, and are one of the most frequent causes of bugs in large systems. (Similar issues are considered in [Rich & Shrobe 1975]).

> **Whoops! I guess that last cleanup should be, "left 90".**

Well, that would work. Another approach would be to change neck, in this case, to do a forward instead of a back.

I like my way better. Anyway, I have to go home now. Goodbye.

SAVING WORK AS: DEBBIE/WORK/32. A PLEASANT DAY TO YOU!

3. Conclusions

The scenario included much detail, but only a few basic motivations guided Sherlock's performance. These were:

1. to encourage good problem solving habits such as modularity, as was illustrated both by the construction of main steps as subprocedures rather than in-line code segments, and by the use of state transparency to facilitate rearrangements;

2. to allow dialogue to take place at the problem solving level, rather than at the level of programming language syntax;

3. to facilitate the use of "evolutionary" plans (plans that take advantage of previously solved problems and programs), as was the case with the use of a previously defined iterative SQUARE program to suggest the pattern for a halting triangle.

4. to guide a process of successive experimentation and debugging, one of the important problem solving techniques that an individual must master.

We began this essay with the claim that the design of AI based personal learning environments is a useful vehicle for pursuing goals of AI, psychology, pedagogy and computer science. The scenario portrays a tutoring program which employs diverse sorts of knowledge: knowledge about problem solving, programming and geometry, as well as knowledge about psychological modelling, discourse pragmatics and pedagogical strategy. In order to construct such computer programs, one must explicitly represent the problem solving methods and domain-specific knowledge to be taught, as well as the pedagogical strategies to be employed. Moreover, the performance of such programs serves as one kind of test of the theories which they embody.

1. Developing a computational understanding of problem solving is a central AI concern. Theories are precisely characterized as high level symbolic computer programs, in a fashion similar to the expression of physical theories in the language of mathematics. The extent of support for an AI theory is determined by the competence and efficiency of the associated computer program in performing its prescribed tasks.

2. Modelling the student's current knowledge, not only about the particular domain, but also about planning and debugging strategies, combines several crucial psychological issues. A cognitive theory expressed as a computer program is supported to the extent to which it can simulate the student's responses, or predict likely difficulties.

3. Procedurally representing tutorial strategies suggests an innovative theoretical framework for educational research. A pedagogical theory expressed as a tutoring program is supported to the extent to which it tends to elicit the desired problem solving behaviors, or responds appropriately to the student's questions. (Teaching methods isolated in the context of designing a computer tutor may also be appropriate for testing and use by human teachers.)

4. Determining techniques for integrating the contributions of various knowledge sources echoes a recurrent computer science theme. The effectiveness and extensibility of alternative approaches provides evidence regarding the design methodology, data structures, and control structures used.

The Sherlock tutor as a whole is only in a preliminary design stage. In other papers, we begin the technical exposition of the individual areas of expertise that it must exhibit. In [Miller & Goldstein 1976a], we provide an overview which clarifies the role of Sherlock in our overall research project. In [Goldstein & Miller 1976b], we formally develop our theory of planning and debugging. In [Miller & Goldstein 1976b], we apply this theory to protocol analysis for a student writing Logo programs. In [Miller & Goldstein 1976d], we consider the complex issues which must be resolved in order to automate the protocol analysis procedure. In [Miller & Goldstein 1976c] we describe a first step toward implementing such a tutorial program: an interactive editor for program writing that embodies our theory of planning and debugging.

4. Notes

1. Although the authors prefer to think of Sherlock as a computer program which has yet to be implemented, one may also read the scenario from a perspective in which Sherlock is a human pedagogue experimenting with a particular syllabus and style of presentation. Many of the same issues are raised.

2. In order to illustrate the various capabilities which such a tutor should exhibit, particularly the knowledge of problem solving strategy which it should embody, the dialogue is presented with the hypothetical tutor in what would be its *most verbose state. This would be reserved for demonstration purposes.* Most students would normally be expected (and probably desire) to work more independently.

3. The problem of recognizing common two-dimensional sketches drawn on a tablet is an interesting and significant research area in its own right. However, a preliminary version of Sherlock, which constrained the student to select a project from a small fixed database of elementary drawings, could still be of value both theoretically and practically. The diversity of projects selected by beginners during the first few weeks of Logo experience is not great: polygons, spirals, trucks, houses, trees, stick figures. Considerable leverage might be obtained even by human Logo teachers if they restricted projects during the first few weeks to fall within one of a few carefully analyzed sequences.

4. One of the virtues of constructing computer based learning environments is that such questions may be resolved experimentally by contrasting slightly different versions of a single system. Otherwise, such an issue might be debated indefinitely on informal, intuitive grounds.

5. The process of comparing the performance of a program with a description of its specifications is known variously as "performance annotation" [Goldstein 1974], "meta-evaluation" [Hewitt & Smith 1975], or simply, "proof of correctness" (a special case) [Floyd 1971]. The astute reader may question the possibility of a computer program analyzing the behavior of another computer program, in view of the "halting problem" and similar negative results in the theory of computation. The apparent paradox is resolved by noting that although such a module cannot be constructed "in general," programs can be devised to handle a wide range of useful special cases. For more details on Mycroft's approach to this problem consult the Goldstein reference.

6. Debbie has just raised a common objection: If Sherlock is to be so sophisticated that it can, in fact, perform "automatic programming" within its limited domain, why should Debbie bother to learn programming? But to argue that, given such a tool, she should not (or would not want to) learn to do it herself, is to argue against competent pedagogy in general. The same argument can be proffered with respect to learning arithmetic given the aid of an electronic calculator, or learning any skill, for that matter, given access to an expert human teacher.

5. References

[Austin 1976]

Austin, Howard, *Teaching Teachers Logo, The Lesley Experiments*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 336 (Logo Memo 23), April 1976.

[Brown et al. 1974]

Brown, John Seely, Richard R. Burton and Alan G. Bell, *SOPHIE: A Sophisticated Instructional Environment for Teaching Electronic Troubleshooting (An Example of AI in CAI)* (Final Report), Bolt, Beranek and Newman, Report 2790 (Artificial Intelligence Report 12), March 1974.

[Burton & Brown 1976]

Burton, Richard R., and John Seely Brown, "A Tutoring and Student Modelling Paradigm for Gaming Environments," in R. Colman and P. Lorton Jr. (eds.), *Computer Science and Education* (Advance Proceedings of the Association for Computing Machinery Special Interest Groups on Computer Science Education and Computer Uses in Education Joint Symposium, Anaheim, Cal.), SIGCSE Bulletin, Volume 8, Number 1 (SIGCUE Topics Volume 2), February 1976, pp. 236-246.

[Collins et al. 1975]

Collins, Allan, Eleanor H. Warnock, Nelleke Aiello and Mark L. Miller, "Reasoning From Incomplete Knowledge," in D. Bobrow and A. Collins (eds.), *Representation and Understanding: Studies in Cognitive Science*, New York, Academic Press, 1975, pp. 383-415.

[Floyd 1971]

Floyd, Robert W., *Toward Interactive Design of Correct Programs*, Proceedings of IFIP Congress 71, North Holland Publishing Co., Amsterdam, Netherlands, August 1971.

[Goldstein, G. 1972]

Goldstein, Gerrianne, *Logo Classes Commentary*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Logo Working Paper 5, February 6, 1972.

[Goldstein 1974]

Goldstein, Ira P., *Understanding Simple Picture Programs*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Technical Report 294, September 1974.

[Goldstein & Miller 1976b]

Goldstein, Ira P., and Mark L. Miller, *Structured Planning and Debugging: A Linguistic Theory of Design*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 387 (Logo Memo 34), December 1976b.

[Hewitt & Smith 1975]

Hewitt, Carl, and Brian Smith, *Towards a Programming Apprentice*, IEEE Transactions on Software Engineering, vol. SE-1, no. 1, March 1975.

[Hildreth 1975]

Hildreth, Ellen, *Logo for First and Second Graders: A Teachers Helper*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Logo Working Paper 40, August 1975.

[Miller & Goldstein 1976a]

Miller, Mark L., and Ira P. Goldstein, *Overview of a Linguistic Theory of Design*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 383 (Logo Memo 30), December 1976a.

[Miller & Goldstein 1976b]

Miller, Mark L., and Ira P. Goldstein, *Parsing Protocols Using Problem Solving Grammars*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 385 (Logo Memo 32), December 1976b.

[Miller & Goldstein 1976c]

Miller, Mark L., and Ira P. Goldstein, *SPADE: A Grammar Based Editor for Planning and Debugging Programs*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 386 (Logo Memo 33), December 1976c.

[Miller & Goldstein 1976d]

Miller, Mark L., and Ira P. Goldstein, *PAZATN: A Linguistic Approach to Automatic Analysis of Elementary Programming Protocols*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 388 (Logo Memo 35), December 1976d.

[Okumura 1972]

Okumura, K., *Logo Classes Commentary* Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Logo Working Paper 6, February 1972.

[Papert 1971a]

Papert, Seymour A., *Teaching Children to be Mathematicians Versus Teaching About Mathematics*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 249, 1971.

[Papert 1971b]

Papert, Seymour A., *Teaching Children Thinking*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 247 (Logo Memo 2), 1971.

[Papert 1973]

Papert, Seymour A., *Uses of Technology to Enhance Education*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 298 (Logo Memo 8), June 1973.

[Piaget 1972]

Piaget, Jean, *The Science of Education and the Psychology of the Child*, New York, Viking Press, 1972.

[Polya 1965]

Polya, George, *Mathematical Discovery* (Volume 2), New York, John Wiley and Sons, 1965.

[Rich & Shrobe 1975]

Rich, Charles, and Howard E. Shrobe, *Understanding LISP Programs: Towards a Programmer's Apprentice* (Master's Thesis), Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, August 1975.

[Solomon 1976a]

Solomon, Cynthia, "Leading a Child to a Computer Culture," in R. Colman and P. Lorton Jr. (eds.), *Computer Science and Education* (Advance Proceedings of the Association for Computing Machinery Special Interest Groups on Computer Science Education and Computer Uses in Education Joint Symposium, Anaheim, Cal.), SIGCSE Bulletin, Volume 8, Number 1 (SIGCUE Topics Volume 2), February 1976, pp. 79-83.

[Solomon 1976b]

Solomon, Cynthia, *A Case Study of a Young Child Doing Turtle Graphics in Logo*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Memo 375, Logo Memo 28, July 1976.